CS 3140 Midterm 1

AJ Nye

TOTAL POINTS

62 / 72

QUESTION 1

Question 1 - Version Control 6 pts

1.1 Multiple Choice 3 / 3

✓ **- 0 pts** *Correct - 1, 2, 5*

- 1 pts 1 correct answer not marked

- 2 pts 2 correct answers not marked

- 3 pts No correct answers marked

- 1 pts 1 incorrect answer marked

- 2 pts 2 incorrect answers marked

1.2 Follow-Up 3 / 3

✓ **- 0 pts** *Correct*

git add Myfile.java
git commit -m "Some commit message"
git push

- 1 pts git add MyFile.java missing or largely incorrect

- 0.5 pts git add didn't include file name

- 0.5 pts git add is out of order (it must be first)

- 1 pts git commit -m "Some commit message" missing or largely incorrect

- 0.5 pts git commit missing commit message, or other minor error

- 0.5 pts git commit out of order (must be after add and before push)

- 1 pts git push missing or very incorrect

- 0.5 pts git push included unnecessary arguments

- 0.5 pts git push out of order (must be the last step)

- 1.5 pts No git keyword

QUESTION 2

Question 2 - Software Quality 5 pts

2.1 Multiple Choice 3 / 3

✓ **- 0 pts** *Correct - 1, 5*

- 1 pts One correct answer not marked

- 2 pts Neither correct answer marked

- 1 pts One incorrect answer marked

- 2 pts Two incorrect answers marked

- 3 pts Three incorrect answers marked

2.2 Follow-up 2 / 2

✓ **- 0 pts** *Correct - External software quality refers to the quality as experienced by the customer with regards to usage, Internal Software quality refers to the quality as seen by the developers with regards to maintainability*

- 1 pts One definition above incorrect, or answer is vague

- 2 pts Incorrect or blank answer

QUESTION 3

Question 3 - Testing 6 pts

3.1 Multiple Choice **3 / 4**

- **0 pts** Correct - 3, 6
- **1 pts** Failed to mark one correct Answer
- **3 pts** Failed to mark either correct Answer

✓ **- 1 pts** *Marked one incorrect answer*

- **2 pts** Marked two incorrect answers
- **4 pts** Marked 3 or more incorrect answers

3.2 Follow-up **1 / 2**

- **0 pts** Correct - unit testing refers to testing individual functions or modules, system testing refers to testing the entire software system as an end-user would - that is, utilizing the system from an external perspective

✓ **- 1 pts** *One definition is incorrect, the other correct*

- **2 pts** Answer is incorrect, too vague, or missing

QUESTION 4

Question 4 - Build Tools 5 pts

4.1 Multiple Choice **4 / 4**

✓ **- 0 pts** *Correct - 1, 4, 5, 6*

- **1 pts** Failed to mark 1 correct answer
- **2 pts** Failed to mark 2 correct answers
- **4 pts** Failed to mark 3 or more correct answers
- **1 pts** Marked 1 incorrect answer
- **3 pts** Marked 2 incorrect answers

4.2 Follow-up **1 / 1**

✓ **- 0 pts** *Correct - build.gradle*

- **1 pts** Anything else - no partial credit for any incorrect name, no matter how close

QUESTION 5

Question 5 - Test Driven Development 9 pts

5.1 Multiple Choice **1 / 3**

- **0 pts** Correct - 1, 4, 5

✓ **- 1 pts** *Failed to mark 1 correct answer*

- **2 pts** Failed to mark 2 correct answers
- **3 pts** Failed to mark all 3 correct answer

✓ **- 1 pts** *Marked 1 incorrect answer*

- **2 pts** Marked 2 incorrect answers

5.2 Equivalence Test **2 / 2**

✓ **- 0 pts** *Correct - an equivalence test, such as timeConverter(7, 30) returns "7:30 a.m." (p.m. times are also considered equivalence cases)*

- **1 pts** Valid equivalence input, but equivalence output is incorrect
- **2 pts** Either an incorrect test case, or this is a boundary or exception case, or used 2:30 p.m. directly (as instructions said you couldn't use it)

5.3 Boundary Test **0 / 2**

- **0 pts** Correct - a correct boundary case, such as timeConverter(0, 0) - "12:00a.m.: (or any time during zero hour), or timeConverter(12, 0) - "12:00 p.m." (anytime during the 12th hour) Times at 11:59p.m. are also acceptable.
- **1 pts** Gave a correct p.m. test case as a boundary test case that was not 12:00p.m. (11:59 p.m. also accepted), such as (13,17) - "1:37 p.m." - Most p.m. cases are equivalence cases.
- **1 pts** Valid boundary input, but equivalence output is incorrect

✓ **- 2 pts** *Incorrect, or gave an a.m. equivalence case or an exception case.*

5.4 Exception Test **2 / 2**

✓ **- 0 pts** *Correct - Exception test case, like timeConverter(-2, 63), where *either* input is invalid.*

For this question, output was not graded

- 2 pts Did not give an exception test case

OR

Gave a syntactically incorrect test case (Java would enforce integer typing at compile time, so you cannot execute this test, and tests require execution)

QUESTION 6

Question 6 - Lambda Body 5 pts

6.1 Multiple Choice **3 / 3**

✓ **- 0 pts** *Correct - 4 and only 4*

- 3 pts Didn't mark the lone correct answer - you had to mark at least the correct answer to get any points

- 1 pts Marked 1 incorrect answer

- 2 pts Marked 2 incorrect answers

- 3 pts Marked 3 or more incorrect answers

6.2 Follow-up **2 / 2**

✓ **- 0 pts** *Correct - functions cannot be stored as variables, nor passed as arguments or returned by functions.*

Saying any one of the above, with no contradictory information, is sufficient.

- 1 pts Partially correct answer, but doesn't mention **explicitly** that functions cannot be treated as values (i.e., they can't be stored in variables, passed as arguments, etc.)

- 2 pts Incorrect answer, or blank

QUESTION 7

Question 7 - Analyzability 2 pts

7.1 First Blank **1 / 1**

✓ **- 0 pts** *Correct - Readability*

- 1 pts Incorrect - note there is no partial credit if Readability and Understandability are out of order

7.2 Second Blank **1 / 1**

✓ **- 0 pts** *Correct - Understandability*

- 1 pts Incorrect - note there is no partial credit if Readability and Understandability are out of order

QUESTION 8

Question 8 - Matching 4 pts

8.1 First Letter **1 / 1**

✓ **- 0 pts** *Correct - F*

- 1 pts Incorrect

8.2 Second Letter **0 / 1**

- 0 pts Correct - A

✓ **- 1 pts** *Incorrect*

8.3 Third Letter **0 / 1**

- 0 pts Correct - B

✓ **- 1 pts** *Incorrect*

8.4 Fourth Letter **1 / 1**

✓ **- 0 pts** *Correct - C*

- 1 pts Incorrect

QUESTION 9

9 Defensive Programming **6 / 6**

✓ **- 0 pts** *Correct - meaningfully identified and threw IllegalArgumentExceptions for all of the following:*

- String is null

- indexA is out of bounds (negative or >= length)

- indexB is out of bounds (negative or >= length)

And otherwise did nothing to change what the function would output for a valid input

- 1 pts Did not correctly identify the `String s` is `null` case as an Exception case.

Remember: An empty string, `""`, is not the same thing as `null`

- 0.5 pts Identified the String s is null (such as an if-statement), but did correctly throw an IllegalArgumentException

- 0.25 pts Identified s is null case, but did so after using `s` in a way that would create a NullPointerException

- 0.25 pts Used s = null or s.equals(null) instead of s == null

- 3 pts Did not correctly identified **both** the indexA and indexB out of bounds Exception case

- 1 pts Identified the case where indexA and indexB were too large, but not negative

OR

Vice versa

- 0.5 pts Identified at least one Out Of Bounds Exception case, but did not correctly throw an IllegalArgumentException.

- 0.25 pts Used `indexA > s.length()` instead of `indexA >= s.length` for finding exception (or any other off-by-one exception)

- 1 pts You changed what the function outputs for equivalence cases - This includes throwing an exception for an equivalence case

- 0.5 pts You changed what the function outputs for a boundary case (such as for indexes 0 or size - 1, or size==1), likely due to bad boolean logic. This includes throwing an exception for a boundary case.

- 1.5 pts Demonstrated a significant misunderstanding of how errors are thrown, including throwing the wrong exception, returning an exception, etc.

- 0.5 pts Exception created as an object, but not thrown

- 0.25 pts One or more Exception throws are missing the new keyword or other minor syntax error.

- 0.25 pts Function shouldn't declare unchecked exceptions in their method signature.

- 0.25 pts Other minor error (see comments)

- 0.25 pts Other minor error (see comments)

- 1 pts Attempted to catch an exception that couldn't exist or occur

- 1.5 pts Incorrect and unnecessary usage of a

try-catch block, such as using a try-catch instead of throwing an exception, or catching exceptions that could not be thrown by the code. Or catching IllegalArgumentException when you are trying to convert other errors to it..

- 3 pts Catches a generic or incorrect exception in only one place, but you are grouping unlike errors (i.e. NullPointerException and IndexOutOfBoundsException) or catching an error that could not be thrown (including IllegalArgumentException). You did not specifically and meaningfully identify the specific failures that could occur. This means you could not add error messages to your code to produce a useful error message for the user with your code for each type of error. A client receiving the IllegalArgumentException could not easily trace why it occurred.

- 6 pts Largely incorrect

OR

incomplete

OR

did not catch any meaningful error

OR

not a clear attempt to use defensive programming (i.e., didn't change the code), or otherwise not worth any points.

QUESTION 10

10 Java Streams **9 / 9**

✓ **- 0 pts** *Correct - anything equivalent to (note that this solution uses lambda bodies, but method captures and/or built-in Stream features (like Comparators.comparing) are also acceptable)*

`*.filter(s -> 'A' == s.charAt(0))*
.sorted((a, b) -> b.length() - a.length())
.limit(3)
.forEach(s -> System.out.println(s)) `

- 1 pts 1 Inappropriate operation included

- 2 pts 2 inappropriate operations included

- 4 pts Three or more inappropriate operations included

- 3 pts No filter operation

- 2 pts Filter operation incorrect - either lambda body produces incorrect filter, or lambda body semantically incorrect, or significant syntactic errors

- 1 pts Syntax error or minor logic error (such as = instead of ==) in lambda body of filter,

- 3 pts No sorted operation

- 2 pts Sorted lambda body doesn't clearly sort based on the size of the string

- 1 pts Sorted sorts in ascending order, not descending order

OR

Sorted has some correct idea but VERY incorrect syntax

- 0.5 pts Sorted is semantically correct, but has some other minor error, such as a minor syntax

error

- 1 pts limit() operation incorrect (missing, incorrect number, or not after sorted) - limit doesn't need to be immediately after sorted, but it must be after sorted

- 2 pts Incorrect terminal operation - should be forEach

If forEach is used, but is not the **last** operation, you get this deduction as well.

- 1 pts Correctly used forEach as terminal operation, but lambda body is semantically incorrect, or very incorrect syntax

- 0.5 pts Correctly used forEach as terminal operation, but minor syntax error

- 2 pts Referenced myList multiple times unneccesarily (this would be a syntax error)

QUESTION 11

JUnit Test Writing 15 pts

11.1 Test for returns True **6 / 7**

- 0 pts Correct

- 0.5 pts Test was not written correctly as a method (i.e. `public void testName() {`) or signature is incorrect in more than 1 way.

- 0.25 pts Test method signature incorrect in one way (such as not returning void, or having any parameters)

- 1 pts Did not meaningfully attempt to setup test VendingMachine instance for test (such as calling the stock and vend methods statically rather than making an instance)

OR

Test Object isn't created due to a serious misunderstanding of how to instantiate the VendingMachine test object (such as only creating a HashMap, and not a vending machine)

- 0.5 pts Incorrectly setup test object for test due to some minor error

- 1.5 pts Did not correctly call the vend method and test the output value (in this case, `true`)

- 1 pts Correctly call the test method, but do not check for correct output value (in this case, `true`)

OR

Correct use asserTrue, but do not call method correct

- 0.5 pts Executes vend more than once in a single test.

- 0.25 pts Incorrect syntax or other minor error on testing the output of the vend method.

✓ **- 1 pts** *Did not test any post-conditions other than the return value*

- 2 pts ONE-TIME-DEDUCTION - this deduction to test 1 is for something incorrect about all of your tests.

Tests are not independently implemented (that is, the results of one test could affect another, or you assume tests are executed "in-order"). Example: using the same instance of the test object for everything method without reseting the value with a Constructor call in @BeforeEach

- 1 pts ONE-TIME-DEDUCTION - this deduction to test 1 is for something incorrect about all of your tests.

Did not include any @Test tags on methods, or only included 1 @Test tag

- 4 pts Only test for return `true` is largely incorrect or blank

- 7 pts All tests are largely incorrect or blank

11.2 Test for returns False **3 / 4**

- 0 pts Correct

- 0.5 pts Test was not written correctly as a method (i.e. `public void testName() {`) or signature is incorrect in more than 1 way.

- 0.25 pts Test method signature incorrect in 1 way (such as not returning void, or having any parameters, or not having parentheses on the function name)

- 1 pts Did not meaningfully attempt to setup the test state VendingMachine instance for test (such as calling the stock and vend methods statically rather than making an instance)

OR

Test Object isn't created due to a likely misunderstanding of how to instantiate the VendingMachine test object (such as instantiating a HashMap, but not a Vending Machine)

- 0.5 pts Incorrectly setup test object for test due to some minor error

- 1.5 pts Did not correctly call the vend method and check that it returns `false`

- 1 pts Tests that the output of vend is `false` but calls the method vend incorrectly.

OR

Correctly gets the output of vend but doesn't assert that it should be false.

- 0.25 pts Incorrect syntax or other minor error on testing the output of the vend method.

- 0.5 pts Executes vend more than once in a single test.

✓ **- 1 pts** *Did not test any post-conditions other than the the return value of `vend`*

- 4 pts Test for `false` is largely incorrect, blank, or missing

11.3 Exception Test **4 / 4**

✓ **- 0 pts** *Correct*

- 1 pts Test was not written correctly as a method (i.e. `public void testName() {`) or signature is incorrect in more than 1 way.

- 0.5 pts Test method signature incorrect in one way (such as not returning void, or having any parameters)

- 1 pts Did not meaningfully attempt to setup test VendingMachine instance for test (such as calling the stock and vend methods statically rather than making an instance)

OR

Test Object isn't created due to a serious misunderstanding of how to instantiate the VendingMachine test object

- 0.5 pts Incorrectly setup test object for test due to some minor error

- 2 pts Did not correctly call the vend method inside an assertThrows to ensure an exception was thrown, or assertThrows usage is not meaningful

- 1 pts Correctly call the test method, but do not check if it throws an Exception in a meaningful way or syntax is signficantly incorrect.

- 0.5 pts Minor incorrect syntax on assertThrows

- 0.5 pts Executes vend more than once in a single test.

- 4 pts Test for Exception is largely incorrect or blank

CS 3140 – Midterm Exam

By filling out your name below, you agree to the following Honor pledge
"On my honor, I pledge that I have neither given nor received help on this exam."

Name: Andrew Nye

Signature: Andrew Nye

Computing ID: pwu5ny

Exam Instructions:

1) Do not open the exam until you are instructed to do so.

2) Once the exam begins, you will have 75 minutes to complete the exam.

3) Write your Computing ID at the top of each page, including the last page. This ensures that if pages somehow get separated, we can still grade your exam.

4) This exam is **closed note, closed laptop.** You should only have a writing utensil with you during the exam. We recommend using pencil rather than pen.

5) When you need to write code, all code should be written using the Java programming language.

6) In order to ensure fairness and an equal exam experience, **the proctors will be unable to answer clarifying questions during the exam.** If you are unsure, you may write your assumptions down on the exam near your answer and we may take them into account. If a question is later deemed misleading or unfair, it will be thrown out and not counted against any student's grade.

7) If you need to use the restroom during the exam, you must leave your phone with the exam proctors. You can retrieve it at the end of the exam period.

8) **DO NOT DETACH ANY PAGES UNLESS THE EXAM PAGE SAYS YOU ARE EXPLICITLY ALLOWED TO.** There is some reference code on the last page of the exam, and you may detach that. However, this *must* be turned in at the end of the exam. Failure to turn in every page will result in a significant deduction.

BEST OF LUCK!

PART 1 : Multiple Choice ++

Answer the following multiple choice questions. Please note that many of these questions may have more than 1 correct answer, and you are expected to select all of them. Please pay attention to the end of the question, as it will say how many answers are possibly correct. Each multiple choice question has a follow-up short answer question.

Question 1) Version Control (6 points) Which of the following are reasons to use version control software like git? Select all the apply **(0-5 answers)**:

1) Allows for multiple members to work on a single project asynchronously
2) Can be a helpful debugging tool by going backwards through commits
3) Git offers automatic conflict resolution so programmers do not have to worry about it
4) Version control will automatically ensure that your code is running correctly Tests have to be manually written to ensure your code is correct, not automatic
5) Version control is a widely used technology in the software industry.

Follow-up: While working in a git repository hosted on Github, you created a new file called "MyFile.java" that you want to add to the **local and remote** repository. Write three git commands, **in order**, that will add the file to both the **local and remote repository. (1 command on each line)**

1) git add MyFile.java
2) git commit -m "Added MyFile.java"
3) git push

Question 2) Software Quality (5 points) - Which of the following are measures of **internal** software quality? (0-4 answers)

1) Maintainability how easy is it to maintain
2) Functionality how well it functions to the design
3) Reliability how often does it fail over a long period of time
4) Usability how easy is it to use
5) Analyzability how readable and understandable the code is

internal: maintain, change, robust, analyze

external: functional, reliable, usable

Follow-up: What is the difference between **internal** and **external** software quality **(1-2 sentences)?**

Internal software quality measures how easy it is for you and other developers to read, maintain and change the software.
External software quality measures how well it runs and functions through the view point of the customer.

Question 3) Testing (6 points) – circle all of the statements below about testing that are accurate. **(1-3 answers)**

1) If a test fails, there **must** be a bug in the code being tested. Assuming the test is written correctly

2) If a test passes, there **cannot** be a bug in the code being tested.

3) The primary purpose of testing is to find defects

4) The primary purpose of testing is to prove code works perfectly on all input

5) Exhaustive testing is the best approach programmers can take when testing ✓ Rarely feasible, so programmers must use equivalence partitioning

6) A test scenario passes when the **expected output** and **actual output** match

Follow-up: Briefly **(1-2 sentences)** Explain the **difference** between **unit** testing and **system** testing:

Unit testing tests one section of the project, while system testing tests how all units run together in the entire system or project. System testing includes unit testing of many units that comprise the system.

Question 4) Build Tools (5 points) – gradle is an example of a build tool. Which of the following things have we had gradle automate in our class homeworks and lectures – circle all the apply **(1-4 answers):**

1) Automatically generate a compiled .jar file with "./gradlew build"

2) Automatically generate comments for our code, including header and method comments

3) Automatically add other people to our project, like team members on homework

4) Automatically run our JUnit tests for a given project Runs when the gradle is built, so it is automatic, won't let us build otherwise

5) Automatically download and install external dependencies, ensuring the correct version number

6) Automatically download and install gradle with "./gradlew build"

Follow-up: What is the name of the file, typically in the root directory of the project where the instructions for gradle build are stored? **(1 file name)**

build.gradle

Make tests based on specification → write code until tests pass

Question 5) (9 points) Which of the following **correctly** describe **Test Driven Development** practices **(0-5 answers)**

1) You must always write a failing test before you are allowed to write new code

2) When writing a function, you should write **every test** you intend for that function before writing any code

3) You should never throw exceptions, because we cannot test for exceptions

4) You write "just enough" code to make your test(s) pass

5) You are primarily using Black-Box testing strategies

Once your tests pass, then the specifications are met, assuming you wrote correct and an adequate amount of tests

Follow-up: Consider a function timeConverter(int hour, int minute) that takes in an hour and minute by 24-hour time and returns a String of the time according to 12 hour time. For example, timeConverter(14, 30) would return "2:30 p.m." Write one one of each kind of test case: **Equivalence, Boundary, and Exception**. You cannot use 2:30 p.m. as an example.

Test type	Input	Expected return value
Equivalence	timeConverter(10, 15)	10:15 AM
Boundary	timeConverter(24, 59)	12:59 AM
Exception	timeConverter(-5, -20)	

Question 6) (5 points) Which of the following lambda bodies could be a **Predicate,** i.e. a lambda body that could be used with the Stream function **filter**. (1-3 answers)

filter (bool)

1) x -> x * 2

2) item -> System.out::println

3) (a, b) -> Integer.compare(a, b) Returns integer

4) x -> x % 2 == 1

5) () -> account.withdraw(300)

myMethod x; is not valid

Follow-up: In Java, we need Functional Interfaces to implement lambda bodies because functions **are not** "first-class citizens". What does that mean in the context of Java and functions?

functions can not be defined and put into variables to be run that way. No variable can store or have an type be a function.

Question 7) Fill in (2 points): When discussing code Analyzability, we mentioned two key terms that make up analyzability. One term refers to being able to correctly interpret **syntax**, while the other term refers to being able to correctly interpret **semantics.** Fill in the blanks with those two terms below.

readability is a necessary, but insufficient condition for understandability

PART 2 – Matching – the following questions involve matching elements on the left with elements on the right. Note that many questions contain "red herrings", that is answers on the right that won't be matched with anything on the left.

Question 8) How Java Works (4 points) Match the components of JAVA on the left (abbreviated) to their purpose on the right. Write the *letter* of the matching definition in the blank to the right of the abbreviation. Note that two answers are red herrings, and will not be used:

Abbreviation	Matching Letter	Purposes
JDK	F	**(A)** An abstract interface that describes instruction for a virtual computing machine
JRE	B	**(B)** A specific implementation of **(A)** that connects with a specific operating system and underlying hardware
JVM	A	**(C)** The part of **(B)** that specifically send instructions to the CPU (Central Processing Unit)
JIT	C	~~**(D)**~~ The part of the Java specifically designed to run JUnit tests, but doesn't affect normal operation
		~~**(E)**~~ The part of Java that is specifically used by IDEs, like Eclipse or Intellij, but cannot be used in any other context
		(F) The part of Java that compiles .java source code files into .class bytecode files.

JVM is abstract, JRE is an implementation of JVM.

The JIT is part of the JVM that covers bytecode into machine readable instructions. It is part of JRE.

PART 3 Code Reading and Writing

Question 9) (6 points) Defensive Programming

The following function takes in a String and two indexes and returns whether or not the character at the two indexes are the same. Rewrite the function using **defensive programming**. Specifically, if any of the arguments result in exceptional behavior, you should throw an IllegalArgumentException (you do not need to include error messages). Do not allow any other exceptions than IllegalArgumentException to be thrown.

```
public static boolean sameLetter(String s, int indexA, int indexB) {
      char letterA = s.charAt(indexA);
      char letterB = s.charAt(indexB);
      if (letterA == letterB) {
            return true;
      }
      return false;
}
```

```
public static boolean sameLetter(String s, int indexA, int indexB){
if (s==null || indexA <0 || indexA >= s.length() || indexB<0 || indexB >= s.length()){
            throw new IllegalArgumentException;
   }
      char letterA = s.charAt(indexA);
      char letterB = s.charAt(indexB);
      if (letterA == letterB){
            return true;
      }
      return false;
}
```

Question 10) (9 points) Streams and Functional Programming.
Write the function described below using **Java streams**. The function takes in a list of strings, and **prints** (using System.out.println) the **three longest strings** that start with the letter **"A"** (specifically, capital A) sorted by length in **descending order**. Example: if myList is:

```
{"Aardvark", "Eagle", "Spaghetti", "Ant", "Aunt", "Apple"}
```

The function should print:

```
Aardvark
Apple
Aunt
```

You may use the intermediate operations functions **map, filter, limit,** and **sorted.** Your last line will be a terminal operation, either: **toList, count,** or **forEach.** I got you started below. You may not need to use all the lines I gave you. Your last line should end with a semicolon. You can either use **one-line** lambda bodies or method captures. You are allowed to use either or both the String methods if needed. **You may or may not use every line below.**

- `boolean startsWith(String s)` - `"catcher".startsWith("cat")` returns **true**
- `char charAt(int index)` - `"catcher".chartAt(2)` returns **'t'**

```
public static void longestAStrings(List<String> myList) {
    myList.stream()
        .filter(s -> s.startsWith("A"))
        .sorted((a,b) -> b.length() - a.length())
        .limit(3)
        .forEach(s -> System.out.println(s));
        .
        .
        .
        .
        .
}
```

Question 11) (16 points) JUnit Test Writing

On the last page of the exam, you will find the class `VendingMachine`. You will be writing a test for the as yet unimplemented function `vend(String candy)`. The block comment above the method gives its specification. On the next page, write 3 JUnit tests.

Assuming that stock("skittles", 0) is valid, when stocking 0 items is allowed

Using Before Each for test vending machine

- -One JUnit test must test when the function returns **true**
- -One JUnit test must test when the function returns **false**
- -One JUnit test must test when the function **throws an Exception.**

All three test cases must follow our rules from lecture and readings on testing instance methods of objects. **You don't need to include the import statements,** but do include any of the necessary @ tags on your test methods. **You are only graded on your tests. Do not implement the method.**

```
public class VendingMachineTest {
    VendingMachine testMachine;

    @BeforeEach
    public void setupTestMachine() {
        testMachine = new VendingMachine();
    }

    @Test
    public void testVendSkittlesInStock() {
        testMachine.stock("skittles", 1);
        assertTrue(testMachine.vend("skittles"));
    }

    @Test
    public void testVendSmartiesOutOfStock() {
        testMachine.stock("smarties", 0);
        assertFalse(testMachine.vend("smarties"));
    }

    @Test
    public void testVendNotStockedMAndMsException() {
        assertThrows(IllegalArgumentException.class, () -> testMachine.vend("m&ms"));
    }
}
```

3

3

Code page: This page may be detached, but must be turned in at the end of the exam.
Question 12 code

```
public class VendingMachine {
	/**
	 * A map of candy names and how many are left in the machine.
	 * Example: if key "Skittles" connects to value 20, then there are 20
	 * packs of Skittles left in the machine.
	 */
	protected Map<String, Integer> stock;

	//Constructor - creates empty vending machine
	public VendingMachine(Map<String, Integer> stock) {
		this.stock = stock;
	}

	//Constructor - creates empty vending machine
	public VendingMachine() {
		this(new HashMap<>());
	}

	/**
	 * Adds candy to vending machine
	 * @param candy - name of the candy
	 * @param count - how many to add to the vending machine
	 */
	public void stock(String candy, int count) {
		//if vending machine has that candy already
		if (stock.containsKey(candy)) {
			//add count to existing total
			stock.put(candy, count + stock.get(candy));
		} else {
			//if vending machine doesn't have that candy
			//add the new candy to the Map
			stock.put(candy, count);
		}
	}

	/**
	 * Attempts to get a particular candy from the vending machine. If there is
	 * at least one of the specified candy, dispense it (reducing the stock of
	 * that candy by 1) and return true. If the candy's stock is zero then return
	 * false (since you can't dispense the candy). However, if the candy is not
	 * stocked in the vending machine at all, throw an IllegalArgumentException
	 * @param candy - the name of the desired candy
	 * @return - true if a candy is available and there is at least 1 in stock,
	 *           false if 0 stock. Throw Exception if candy isn't stocked at all.
	 */
	public boolean vend(String candy) {
		return false; // TODO: Stub - you are not being asked to implement this method
	}

}
```

Assuming you're allowed to stock 0 to show out of stock

You may use the back of this page as scratch paper. However, nothing on this page (front or back) can be graded for any reason.